FPGA IMPLEMENTATION OF AN EFFICIENT PROPORTIONATE AFFINE PROJECTION ALGORITHM FOR ECHO CANCELLATION

Cristian Stanciu[†], *Cristian Anghel*[†], *Constantin Paleologu*[†], *Jacob Benesty*[‡], *Felix Albu*[†], *and Silviu Ciochină*[†]

[†]Telecommunications Department, University Politehnica of Bucharest, Romania
email: {cristian,canghel,pale,felix,silviu}@comm.pub.ro
[‡]INRS-EMT, University of Quebec, Montreal, Canada
email: benesty@emt.inrs.ca

ABSTRACT

This paper presents a field-programmable gate array (FPGA) implementation of a low-complexity proportionate affine projection algorithm (PAPA), in the context of echo cancellation. The proposed algorithm results as a combination between the recently developed "memory"-improved PAPA (MIPAPA) and a dichotomous coordinate descent (DCD) method. The MIPAPA takes into account the "history" of the proportionate factors, which is not the case for the classical PAPAs. Moreover, it achieves a lower computational complexity due to a recursive implementation of the "proportionate history;" as a consequence, the matrix that has to be inverted within the MIPAPA has the time-shift property, which can also lower the complexity. Concerning this task, the DCD technique efficiently performs the matrix inversion, requiring only addition operations. The proposed hardware implementation scheme takes advantage of the algorithm's specific features. The overall performance of the proposed DCD-MIPAPA indicates that it can be a reliable choice for real-world echo cancellation scenarios.

1. INTRODUCTION

Proportionate-type adaptive algorithms were mainly developed in the context of echo cancellation [1]. They are designed to exploit the sparseness character of the echo path, i.e., a certain percentage of the impulse response components have a significant magnitude while the rest are zero or small. The basic idea behind these algorithms is to update each coefficient of the filter independently of the others, by adjusting the adaptation step-size in proportion to the magnitude of the estimated filter coefficient [2], [3]. In this manner, the adaptation gain is "proportionately" redistributed among all the coefficients, emphasizing the large ones in order to speed up their convergence. Consequently, proportionate-type algorithms achieve faster convergence and tracking as compared to the conventional algorithms.

The proportionate normalized least-mean-square (PNLMS) algorithm [2] proposed by Duttweiler almost a decade ago, was one of the first proportionate-type algorithms and maybe the most referred one. The equations used to calculate the step-size control factors of the PNLMS algorithm are not based on any optimization criteria but are designed in an ad-hoc way. For to this reason, after an initial fast convergence phase, the convergence rate of the PNLMS algorithm significantly slows down. Also, it is sensitive to the sparseness degree of the system, i.e., the convergence rate is reduced in the case of dispersive echo paths. In order to deal with these problems, several versions (and combinations) of the PNLMS algorithm were developed, e.g., [4], [5], [6], [7], [8] and the references therein. Among these, the improved PNLMS (IPNLMS) algorithm proposed in [4] is one of the most interesting and practical solutions, mainly due to its simplicity and robustness to the sparseness degree of the echo path.

The affine projection algorithm (APA) [9] was found to be a very attractive choice for echo cancellation. The main advantage of this algorithm over the well-known normalized least-mean-square (NLMS) algorithm consists of a superior convergence rate, especially for speech signals. Since APA is a generalization of the NLMS algorithm, the "proportionate" idea was straightforwardly extended and several proportionate APAs (PAPAs) were developed, e.g., [10], [11], [12], [13]. Among these, the recently proposed "memory"-improved PAPA (MIPAPA) [13] is very appealing in terms of both convergence performance and computational complexity. The MIPAPA takes into account the "history" of the proportionate factors, which is not the case for the classical PAPAs. Consequently, this algorithm outperforms its classical counterparts, achieving both faster tracking and lower misadjustment. Moreover, it has a lower computational complexity due to a recursive implementation of the "proportionate history;" besides, the matrix that has to be inverted within the algorithm has the time-shift property, which can also lower the complexity.

However, the matrix inversion remains the most difficult operation of the APAs/PAPAs, mainly in terms of number of multiplications. In order to overcome this issue, the dichotomous coordinate descent (DCD) method [14], [15], [16], [17] represents a very attractive choice. This is an iterative technique used to efficiently solve a system of equations, which is a challenging problem encountered within several adaptive algorithms.

In this paper, we propose the DCD-MIPAPA, which results as a combination between the MIPAPA and the DCD algorithm with a leading element [16], [17]. The proposed algorithm inherits the nice features of the MIPAPA, together with the lower complexity provided by the DCD technique, which efficiently performs the matrix inversion requiring only addition operations. In order to validate the performance of the DCD-MIPAPA, we developed a field-programmable gate array (FPGA) implementation, which takes advantage of the algorithm's specific features.

The paper is organized as follows. In Section 2, we developed the DCD-MIPAPA, outlining its advantages over the classical PAPAs. The proposed hardware implementation scheme of the algorithm is detailed in Section 3. Simulations performed in the context of network echo cancellation are provided in Section 4. Finally, Section 5 concludes this work.

2. THE DCD-MIPAPA

Consider an adaptive filter of length L, defined by the coefficients vector $\hat{\mathbf{h}}(n) = [\hat{h}_0(n),\ \hat{h}_1(n),\ \ldots,\ \hat{h}_{L-1}(n)]^T$, where n is the discrete-time index and superscript ^{T} denotes transposition. The equations that define the classical PAPAs are [10], [11], [12]:

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{X}^T(n)\hat{\mathbf{h}}(n-1), \quad (1)$$

$$\mathbf{P}(n) = \mathbf{G}(n-1)\mathbf{X}(n), \quad (2)$$

$$\mathbf{M}(n) = \delta \mathbf{I}_P + \mathbf{X}^T(n)\mathbf{P}(n), \quad (3)$$

$$\hat{\mathbf{h}}(n) = \hat{\mathbf{h}}(n-1) + \alpha \mathbf{P}(n)\mathbf{M}^{-1}(n)\mathbf{e}(n), \quad (4)$$

where $\mathbf{d}(n) = [d(n), d(n-1), \ldots, d(n-P+1)]^T$ is a vector containing the most recent P samples of the desired signal (with P denoting the projection order), the matrix $\mathbf{X}(n) = [\mathbf{x}(n), \mathbf{x}(n-1), \ldots, \mathbf{x}(n-P+1)]$ is the input signal matrix, with $\mathbf{x}(n-p) =$

$[x(n-p),x(n-p-1),\ldots,x(n-p-L+1)]^T$ being the input signal vectors ($p=0,1,\ldots,P-1$), $\mathbf{G}(n-1)=\text{diag}[g_0(n-1),g_1(n-1),\ldots,g_{L-1}(n-1)]$ is an $L\times L$ diagonal matrix that assigns an individual step size to each filter coefficient (in order to "proportionate" the algorithm behavior), the constant α denotes the step-size parameter, δ is the regularization constant, and $\mathbf{I}_P$ is the $P\times P$ identity matrix. Clearly, when the "proportionate" matrix is chosen as $\mathbf{G}(n-1)=\mathbf{I}_L$ (where $\mathbf{I}_L$ is the $L\times L$ identity matrix), the classical APA [9] is obtained.

In practice, the matrix product from (2) is evaluated by taking into account the diagonal character of the matrix $\mathbf{G}(n-1)$, i.e.,

$$\mathbf{P}(n)=[\mathbf{g}(n-1)\odot\mathbf{x}(n),\ldots,\mathbf{g}(n-1)\odot\mathbf{x}(n-P+1)], \quad (5)$$

where $\mathbf{g}(n-1)=[g_0(n-1),g_1(n-1),\ldots,g_{L-1}(n-1)]^T$ is a vector containing the diagonal elements of $\mathbf{G}(n-1)$ and the operator $\odot$ denotes the Hadamard product, i.e., $\mathbf{a}\odot\mathbf{b}=[a_0b_0,a_1b_1,\ldots,a_{L-1}b_{L-1}]^T$, with $\mathbf{a}$ and $\mathbf{b}$ being two vectors of length L.

It is known that the APA can be viewed as an algorithm with "memory," i.e., it takes into account the "history" of the last P time samples. However, the classical PAPAs do not take into account the "proportionate history" of each coefficient $\hat{h}_l(n-1)$, with $l=0,1,\ldots,L-1$, but only its proportionate factor from the current time sample, i.e., $g_l(n-1)$. Motivated by this limitation, the algorithm proposed in [13] takes advantage of the "proportionate memory" of the algorithm, by choosing the matrix

$$\mathbf{P}'(n)=[\mathbf{g}(n-1)\odot\mathbf{x}(n),\ldots,\mathbf{g}(n-P)\odot\mathbf{x}(n-P+1)], \quad (6)$$

instead of the matrix $\mathbf{P}(n)$ from (5). Since it was developed using the proportionate factors of the IPNLMS algorithm [4], i.e.,

$$g_l(n-1)=\frac{1-\kappa}{2L}+(1+\kappa)\frac{|\hat{h}_l(n-1)|}{2\sum_{l=0}^{L-1}|\hat{h}_l(n-1)|},\ 0\le l\le L-1, \quad (7)$$

where κ ($-1\le\kappa<1$) is a parameter that controls the amount of proportionality, the algorithm proposed in [13] was called the "memory"-improved PAPA (MIPAPA).

The advantage of this modification is threefold. First, the MIPAPA takes into account the "history" of the proportionate factors from the last P steps. Of course, the gain in terms of fast convergence and tracking becomes more apparent when the projection order P increases.

Second, the matrix in (6) can be realized recursively as

$$\mathbf{P}'(n)=\begin{bmatrix}\mathbf{g}(n-1)\odot\mathbf{x}(n) & \mathbf{P}'_{-1}(n-1)\end{bmatrix}, \quad (8)$$

where the matrix $\mathbf{P}'_{-1}(n-1)=[\mathbf{g}(n-2)\odot\mathbf{x}(n-1),\ldots,\mathbf{g}(n-P)\odot\mathbf{x}(n-P+1)]$ contains the first $P-1$ columns of $\mathbf{P}'(n-1)$. Thus, the columns from 1 to $P-1$ of the matrix $\mathbf{P}'(n-1)$ can be used directly for computing the matrix $\mathbf{P}'(n)$. This is not the case in the classical PAPAs, where all the columns of $\mathbf{P}(n)$ [see (5)] have to be evaluated at each iteration, because all of them are multiplied with the same vector $\mathbf{g}(n-1)$. Consequently, the evaluation of $\mathbf{P}(n)$ from (5) needs PL multiplications, while the evaluation of $\mathbf{P}'(n)$ [see (8)] requires only L multiplications.

Third, due to its time-shift property, the matrix to be inverted within the MIPAPA, i.e., $\mathbf{M}'(n)=\delta\mathbf{I}_P+\mathbf{X}^T(n)\mathbf{P}'(n)$, can be recursively evaluated in a very efficient manner as [3]

$$\mathbf{M}'(n)=\begin{bmatrix}\delta+\mathbf{x}^T(n)\mathbf{x}_\mathbf{g}(n) & \mathbf{x}^T(n)\mathbf{P}'_{-1}(n-1)\\ \mathbf{X}^T_{-1}(n-1)\mathbf{x}_\mathbf{g}(n) & \mathbf{M}_{P-1}(n-1)\end{bmatrix}, \quad (9)$$

where $\mathbf{x}_\mathbf{g}(n)=\mathbf{g}(n-1)\odot\mathbf{x}(n)$, the matrix $\mathbf{M}'_{P-1}(n-1)$ contains the first $P-1$ columns and $P-1$ rows of the matrix $\mathbf{M}'(n-1)$ [i.e., the top-left $(P-1)\times(P-1)$ submatrix of $\mathbf{M}'(n-1)$], and the matrix $\mathbf{X}_{-1}(n-1)$ contains the first $P-1$ columns of the matrix $\mathbf{X}(n-1)$. Consequently, only the first row and the first column of $\mathbf{M}(n)$ need to be computed, requiring $2PL-L$ multiplications and $2P(L-1)-L+2$ additions. Moreover, we can further simplify the update (9) by "forcing" the symmetry of the matrix to be inverted [18], i.e., the elements of its first row and column are given by $\mathbf{x}^T(n)\mathbf{P}'_{-1}(n-1)$, then adding δ to the first element. The experiments reported in [18] proved that the effects of this approximation over the MIPAPA's performance are very minor. On the other hand, it can be noticed that the matrix $\mathbf{M}(n)$ from (3) is symmetric but does not have a time-shift character. Consequently, we need to compute all its elements above (and including) the main diagonal, requiring $P^2L/2+PL/2$ multiplications and $P^2(L-1)/2+P(L+1)/2$ additions.

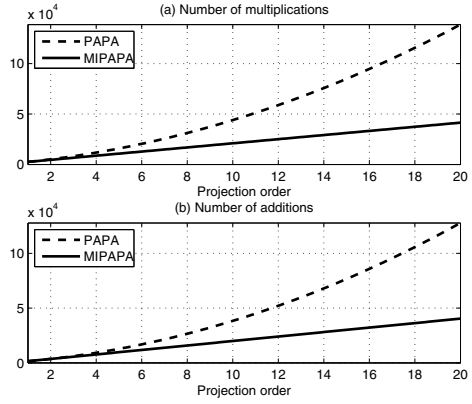

Figure 1: (a) Number of multiplications of the classical PAPA and MIPAPA, as a function of the projection order. (b) Number of additions of the classical PAPA and MIPAPA, as a function of the projection order. The length of the adaptive filter is $L=512$. The complexities associated with the evaluation of the proportionate factors and the matrix inversion were not considered.

A comparison between the computational complexities of the classical PAPAs and MIPAPA is given in Fig. 1, for a filter of length $L=512$. We do not include here the complexities associated with the evaluation of $\mathbf{g}(n-1)$ and the matrix inversion. It is clear that the MIPAPA is much more computationally efficient as compared to the classical version. Also, this advantage becomes more apparent when the projection order P increases. Nevertheless, we still face the "difficult" matrix inversion within the MIPAPA. In order to overcome this issue, let us rewrite the update of the MIPAPA as

$$\hat{\mathbf{h}}(n)=\hat{\mathbf{h}}(n-1)+\alpha\mathbf{P}'(n)\mathbf{s}(n), \quad (10)$$

where $\mathbf{s}(n)$ is the solution of the linear system of P equations

$$\mathbf{M}'(n)\mathbf{s}(n)=\mathbf{e}(n). \quad (11)$$

The main problem is how to obtain the solution $\mathbf{s}(n)$ without evaluating the inverse of the matrix $\mathbf{M}'(n)$. In order to accomplish this task, we can use an inexact line search method known as the DCD with a leading element [16], [17] which allows to iteratively solve the system (11). This algorithm is summarized in Table 1, where the time index n was omitted for clarity. The following notations are used: r_l is the l-th element of the vector $\mathbf{r}$, $\mathbf{M}'_{l,l}$ denotes the l-th diagonal element of the matrix $\mathbf{M}'(n)$, $\mathbf{M}'_{:,l}$ is the l-th column of the matrix $\mathbf{M}'(n)$, and $\text{sign}(\cdot)$ denotes the signum function. The step-size η depends on the value of the parameter H, which also determines the expected amplitude range $[-H,H]$ of the elements of the solution vector $\mathbf{s}$ (represented using M_b+1 bits in fixed-point format). The step-size takes one of predefined M_b values, starting with the initial value H. The algorithm starts updating the elements

Table 1: The DCD Algorithm with a Leading Element

Initialization:	$\mathbf{s}=\mathbf{0}$, $\mathbf{r}=\mathbf{e}$, $\eta=H$, $m=1$	
	For $k=1,\ldots,N_u$	
step 1:	$l=\arg\max_{q=1,\ldots,P}\{\lvert r_q\rvert\}$	($P-1$ additions)
step 2:	$\eta=\eta/2,\ m=m+1$	
step 3:	if $m>M_b$ the algorithm stops	
step 4:	if $\lvert r_l\rvert<(\eta/2)\mathbf{M}_{l,l}$ go to step 2	(1 addition)
step 5:	$\mathbf{s}_l=\mathbf{s}_l+\text{sign}(r_l)\eta$	(1 addition)
step 6:	$\mathbf{r}=\mathbf{r}-\text{sign}(r_l)\eta\mathbf{M}_{:,l}$	(P additions)
	End	

of $\mathbf{s}$ with the most significant bits; it continues with the least significant bits as long as the value of k is not higher than the value of parameter N_u (i.e., the number of maximum allowed updates).

This DCD algorithm searches for the l-th leading element in the residual vector $\mathbf{r}$ [which is initialized with the values of $\mathbf{e}(n)$] that has the highest absolute value at a given iteration (step 1). If the condition in step 4 is satisfied, then the step-size is divided by 2 and the bit with the next lower significance is selected for a possible update (step 2). The condition in step 4 is tested until it is not satisfied and an update is made to the l-th element of the solution vector $\mathbf{s}$ (step 5) and to the entire residual vector $\mathbf{r}$ (step 6). Then, the search for finding a new leading element begins again. The DCD with a leading element stops when the value of k is greater than N_u (i.e., the number of allowed updates has been made and $m<M_b$, thus the condition in step 3 is never satisfied) or the value of m is greater than M_b (i.e., all the bits used for representation of the elements of $\mathbf{s}$ have been determined and $k<N_u$).

More importantly, the DCD algorithm with a leading element requires only addition operations. If H is chosen as a power of 2, the multiplications and divisions in Table 1 are actually bit-shifts. The worst case scenario happens when the algorithm stops because the update corresponding to $k=N_u$ has been made. This means that the largest possible number of operations is $(2P+1)N_u+M_b$ additions. Most of the arithmetic complexity is given by the "successful" iterations, when the condition in step 4 is not satisfied.

Finally, by incorporating the solution for $\mathbf{s}(n)$ (provided by the DCD algorithm) in the update (10), the proposed DCD-MIPAPA results.

3. THE HARDWARE IMPLEMENTATION SCHEME

In this section, we present the implementation of the proposed DCD-MIPAPA on a Xilinx Virtex 5 FPGA, i.e., the XC5VFX70 chip [19], [20], [21]. The length of the adaptive filter is $L=512$ and the projection order of DCD-MIPAPA is set to $P=8$. The system clock frequency is approximately 200 MHz and the sampling frequency is 8 kHz; consequently, there are approximately 25000 clock periods available between two successive samples. This observation allows us to increase the reuse factor for each module.

First, we briefly detail the block scheme of the proposed implementation. Then, a few essential and challenging issues are presented, i.e., the memory usage, the multipliers, the fractional dividers, and the matrix inversion. Also, we discuss the modularity of the FPGA implementation, which is a very important aspect in practice. The functionality of the implementation is verified with Modelsim 6.5 [19], [20].

3.1 The Block Scheme of the DCD-MIPAPA

The block scheme of the proposed implementation is presented in Fig. 2. It can be noticed that the inputs of this scheme are the far-end signal (8-bits per sample), the desired signal (16-bits per sample), the CLOCK (1-bit, with the planned frequency of 200 MHz), and the RESET (which brings the entire algorithm to the initial state). The first two operations in the algorithm are made during the same clock cycles (parallel operation), i.e., the output sample $y(n)=\hat{\mathbf{h}}^T(n-1)\mathbf{x}(n)$ and the sum of the absolute values of the

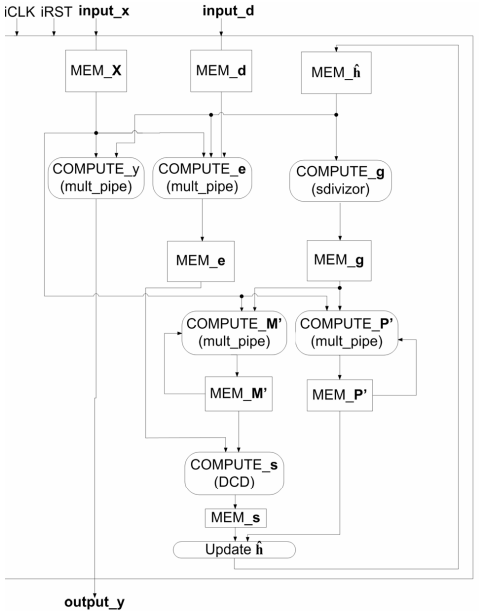

Figure 2: Implementation scheme of the DCD-MIPAPA.

filter coefficients $\sum_{l=0}^{L-1}|\hat{h}_l(n-1)|$ (that will be used for computing the proportionate factors). Next, the proportionate factors [i.e., $g_l(n-1)$, with $l=0,1,\ldots,L-1$; see (7)] are computed, requiring the only L divisions of the algorithm; also, in the same step, the error vector $\mathbf{e}(n)$ [see (1)] is evaluated. In the next two steps, the matrices $\mathbf{P}'(n)$ and $\mathbf{M}'(n)$ are computed; as we have shown in the previous section, both matrices are updated in a recursive manner [see (8) and (9)], thus achieving important computational savings. The most challenging part of the algorithm is the solution of the linear system of equations related with the matrix inversion. This task is accomplish using the DCD algorithm with a leading element (see Table 1) for an $P\times P$ (i.e., 8×8) matrix, $\mathbf{M}'(n)$. Finally, the last step of the DCD-MIPAPA is the update of the filter coefficients, according to (10).

3.2 The Memory Blocks

The FPGA implementation of the DCD-MIPAPA requires a few RAM memory blocks, as follows. The first one is associated with the input signal samples (8-bits format); this memory can be viewed as an $L\times P$ matrix, i.e., 512×8 in our case. Apparently, there is a need for $512\times 8=4096$ memory locations. However, it can be easily noticed that the columns of this matrix have the time-shift property. Therefore, the allocated memory for $\mathbf{X}(n)$ needs only $L+P-1$ locations (i.e., 519 in our case), which represents an important reduction in terms of the memory resources usage.

Another RAM block is associated with the proportionate factors, i.e., $g_l(n-1)$, with $l=0,1,\ldots,L-1$. Due to the efficient update of the matrix $\mathbf{P}'(n)$ [see (8)], only the current proportionate factors need to be stored. Consequently, this memory has 512 locations of 16 bits each.

Finally, the other necessary memory blocks are associated with 1) the filter coefficients (i.e., 512 locations of 16 bits each), 2) the matrix $\mathbf{P}'(n)$ (i.e., $L\times P=512\times 8=4096$ locations), and 3) the matrix $\mathbf{M}'(n)$ (i.e., $P\times P=8\times 8$ locations). Also, a small amount of memory space is used for the error signal vector $\mathbf{e}(n)$, which has $P=8$ locations of 16 bits each.

3.3 The Multipliers

Within the implemented DCD-MIPAPA, all the multiplications are performed in a pipeline manner, i.e., in series of consecutive $L=512$ computations. During each clock cycle, a pair of input factors

is introduced in the pipeline multiplier, which transfers them between a series of cells that shift the binary values and compute a partial result. The result of a complete series of 512 multiplications is available after $512+N_b$ clock cycles, where N_b is the number of bits corresponding to one of the input factors. Taking into account that the multiplications are performed between 16 bits and 8 or 16 bits operands, then N_b is chosen as 8 or 16 (e.g., $N_b=8$ when the input of the multiplier contains the samples of the input signal). Also, N_b represents the number of computational cells contained (in each case) by the pipeline multiplier block.

It is important to outline that some of the multiplications were avoided by using an appropriate step-size α. The value of this parameter is set to $\alpha=2^{-3}+2^{-4}=0.1875$. This means that in the final step of the algorithm (i.e., the update of the filter coefficients) we save 512 multiplications, which are replaced by $2\times512=1024$ bit-shifts and 512 additions.

3.4 The Fractional Divider

The only divisions required within the DCD-MIPAPA are associated with the computation of the proportionate factors [i.e., $L=512$ divisions; see (7)]. We need to take into account that the dividend has to be less or equal to the divisor. However, in our case, this condition is automatically fulfilled because the dividends are the absolute values of the filter coefficients and the divisor is the sum of the absolute values of the filter coefficients. The input values have a 25-bits representation (the dividends are extended to match the length of the divisor) and the result is obtained after 25 clock cycles (equal to the number of bits of the input values). The division by zero is avoided by adding a small positive value ε to the denominator.

Besides the multiplications and the simpler additions and bit-shifts, the divisions are the only required operations within the implementation of the DCD-MIPAPA. Let us remind the fact that the matrix inversion is performed using only addition operations; this is due to the nice features of the DCD algorithm with a leading element [16], [17].

3.5 The DCD Algorithm

The implementation of the DCD algorithm is described in Table 1. In order to have a fix processing delay [independent of the matrix $\mathbf{M}'(n)$], we allocate the same number of clock periods for all the numbers of maximum updates N_u, either if they will be used for real processing or not. More precisely, for each update, the maximum absolute value of the vector $\mathbf{r}$ is found in P clock periods (if still needed). Then, the condition from step 4 is verified. If it is true, the step 2 is run in one clock period, else no action is made; then, P clock periods are dedicated for step 5 and step 6, which can be run in parallel if the condition from step 4 is false, or no action will be performed during this time period if the condition is true. Figure 3 depicts the timing of the algorithm for each performed update. The processing duration is $(2P+1)N_u$ clock periods. The flag sNoAct is set to 1 when $m>M_b$. In this way, it indicates that no action is performed anymore and the DCD algorithm will run without changes just to complete the fix processing duration.

4. SIMULATION RESULTS

Simulations were performed in the context of network echo cancellation. The echo path is the first one from G168 Recommendation [22], padded with zeroes up to 512 coefficients; the sampling rate is 8 kHz. As we have mentioned in the previous section, the length of the adaptive filter is $L=512$ and the projection order is $P=8$. The far-end signal (i.e., the input signal) is either a white Gaussian signal or a speech sequence. The output of the echo path is corrupted by an independent white Gaussian noise (i.e., the background noise at the near-end) with 25 dB echo-to-noise ratio (ENR). The performance measure is the normalized misalignment (in dB) defined as $20\log_{10}\left[\left\|\mathbf{h}-\hat{\mathbf{h}}(n)\right\|_2/\left\|\mathbf{h}\right\|_2\right]$. The results are averaged over 20 independent trials. All the following results were obtained on a finite

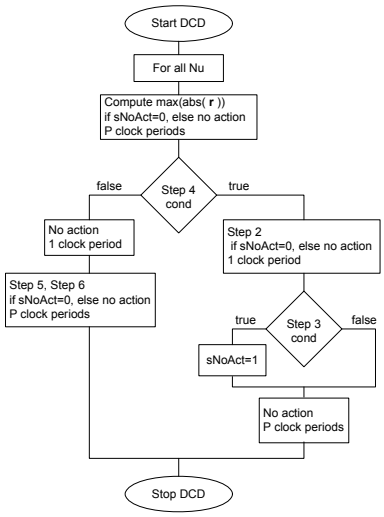

Figure 3: The timing scheme of the DCD algorithm.

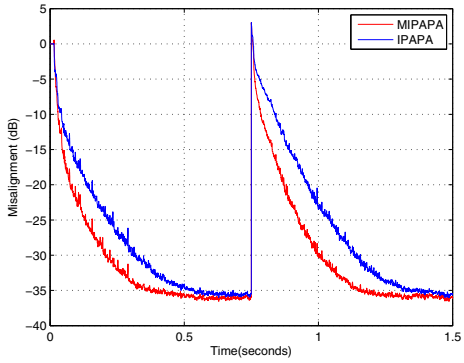

Figure 4: Misalignment of the IPAPA and MIPAPA (infinite precision). The input signal is white and Gaussian, $L=512$, $P=8$, and ENR = 25 dB. The echo path changes at time 0.75 seconds.

precision Matlab simulator, which completely emulates the VHDL implementation behavior.

In all the experiments, the step-size parameter is set to $\alpha=0.1875$ (the reason behind choosing this value was explained in the previous section) and the regularization constant is $\delta=20\sigma_x^2/L$ [23], where σ_x^2 is the variance of the input signal (evaluated on the last L input samples). The parameter that controls the amount of proportionality [4] is set to $\kappa=0$ [see (7)]. In the first experiment, we compare the standard form of the MIPAPA (i.e., using the "classical" matrix inversion and infinite precision) with its original counterpart, i.e., the improved proportionate APA (IPAPA) [11]. The input signal is a white Gaussian noise. In order to evaluate the tracking capabilities of the algorithms, the echo path changes after 0.75 seconds by shifting its impulse response to the right with 20 samples. It can be noticed from Fig. 4 that the MIPAPA outperforms the IPAPA in terms of both fast convergence and tracking. The second simulation compares the standard MIPAPA with the finite-precision DCD-MIPAPA using different values of the parameter N_u (i.e., the number of "successful" iterations). The other parameters of the DCD-MIPAPA are set to $H=128$ and $M_b=14$ (see Table 1), and the conditions are the same as in the previous experiment. It can be noticed from Fig. 5 that the performance of the DCD-MIPAPA is improved for a higher value of N_u. However, a value of $N_u=15$ is sufficient to get a similar behavior with the standard MIPAPA.

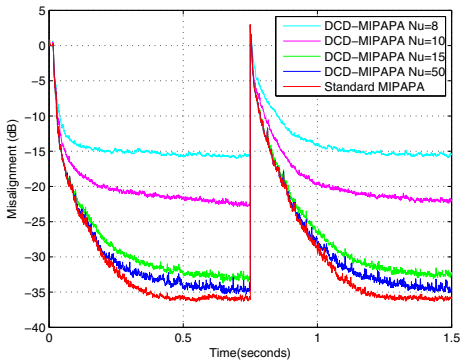

Figure 5: Misalignment of the standard MIPAPA (infinite precision) and DCD-MIPAPA (finite precision) using different values of the N_u parameter. The other conditions are the same as in Fig. 3.

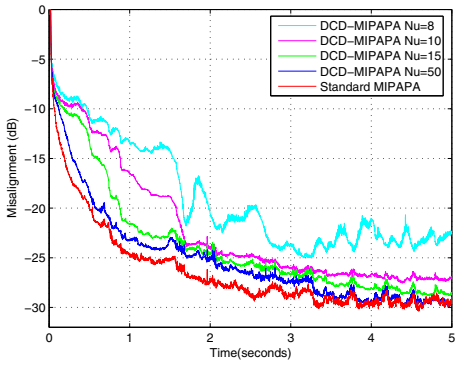

Figure 6: Misalignment of the standard MIPAPA (infinite precision) and DCD-MIPAPA (finite precision) using different values of the N_u parameter. The input signal is speech, $L = 512$, $P = 8$, and ENR $= 25$ dB.

Finally, the last experiment is performed using a speech signal as input (Fig. 6). The conclusions are basically the same as in the previous experiment, i.e., the higher is the value of the parameter N_u, the better is the performance of the DCD-MIPAPA. However, a very high value of N_u (e.g., $N_u > 50$) is not justified, in terms of the compromise between the computational complexity and the convergence performance.

5. CONCLUSIONS

In this paper, we have proposed the DCD-MIPAPA for echo cancellation. This algorithm results as a combination between the recently proposed MIPAPA and the DCD algorithm with a leading element (which is used to efficiently compute the matrix inversion). The provided FPGA implementation represents a low-cost solution, which could be very attractive for real-world echo cancellation scenarios.

Acknowledgment

This work was supported under the Grant POSDRU/89/1.5/S/62557 and Grant UEFISCDI PN-II-RU-TE no. 7/5.08.2010.

REFERENCES

[1] J. Benesty, T. Gaensler, D. R. Morgan, M. M. Sondhi, and S. L. Gay, *Advances in Network and Acoustic Echo Cancellation*. Berlin, Germany: Springer-Verlag, 2001.

[2] D. L. Duttweiler, "Proportionate normalized least-mean-squares adaptation in echo cancelers," *IEEE Trans. Speech, Audio Processing*, vol. 8, pp. 508–518, Sept. 2000.

[3] C. Paleologu, J. Benesty, and S. Ciochină, *Sparse Adaptive Filters for Echo Cancellation*. Morgan & Claypool Publishers, 2010.

[4] J. Benesty and S. L. Gay, "An improved PNLMS algorithm," in *Proc. IEEE ICASSP*, 2002, pp. 1881–1884.

[5] H. Deng and M. Doroslovački, "Proportionate adaptive algorithms for network echo cancellation," *IEEE Trans. Signal Processing*, vol. 54, pp. 1794–1803, May 2006.

[6] J. Arenas-Garcia and A. R. Figueiras-Vidal, "Adaptive combination of proportionate filters for sparse echo cancellation," *IEEE Trans. Audio, Speech, Audio Processing*, vol. 17, pp. 1087–1098, Aug. 2009.

[7] P. Loganathan, A. W. H. Khong, and P. A. Naylor, "A class of sparseness-controlled algorithms for echo cancellation," *IEEE Trans. Audio, Speech, Language Processing*, vol. 17, pp. 1591–1601, Nov. 2009.

[8] F. das Chagas de Souza, O. J. Tobias, R. Seara, and D. R. Morgan, "A PNLMS algorithm with individual activation factors," *IEEE Trans. Signal Processing*, vol. 58, pp. 2036–2047, Apr. 2010.

[9] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," *Electron. Commun. Japan*, vol. 67-A, pp. 19–27, May 1984.

[10] T. Gaensler, S. L. Gay, M. M. Sondhi, and J. Benesty, "Double-talk robust fast converging algorithms for network echo cancellation," *IEEE Trans. Speech, Audio Processing*, vol. 8, pp. 656–663, Nov. 2000.

[11] O. Hoshuyama, R. A. Goubran, and A. Sugiyama, "A generalized proportionate variable step-size algorithm for fast changing acoustic environments," in *Proc. IEEE ICASSP*, 2004, pp. IV-161–IV-164.

[12] S. Werner, J. A. Apolinário Jr., and P. S. R. Diniz, "Set-membership proportionate affine projection algorithms," *EURASIP J. Audio, Speech, Music Processing*, vol. 2007, no. 1, pp. 1–10, 2007.

[13] C. Paleologu, S. Ciochină, and J. Benesty, "An efficient proportionate affine projection algorithm for echo cancellation," *IEEE Signal Processing Lett.*, vol. 17, pp. 165–168, Feb. 2010.

[14] Y. V. Zakharov and T. C. Tozer, "Multiplication-free iterative algorithm for LS problem," *IEE Electronics Lett.*, vol. 40, pp. 567–569, Apr. 2004.

[15] Y. V. Zakharov and F. Albu, "Coordinate descent iterations in fast affine projection algorithm," *IEEE Signal Processing Lett.*, vol. 12, pp. 353–356, May 2005.

[16] Y. V. Zakharov, "Low complexity implementation of the affine projection algorithm," *IEEE Signal Processing Lett.*, vol. 15, pp. 557-560, 2008.

[17] J. Liu, Y. V. Zakharov, and B. Weaver, "Architecture and FPGA design of dichotomous coordinate descent algorithms," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 56, pp. 2425–2438, Nov. 2009.

[18] F. Albu, C. Paleologu, J. Benesty, and S. Ciochină, "A low complexity proportionate affine projection algorithm for echo cancellation," in *Proc. EUSIPCO*, 2010, pp. 6–10.

[19] "Xilinx Virtex 5 family user guide," www.xilinx.com.

[20] "Xilinx ML507 evaluation platform user guide," www.xilinx.com

[21] C. Anghel, C. Paleologu, J. Benesty, and S. Ciochină, "FPGA implementation of a variable step-size affine projection algorithm for acoustic echo cancellation," in *Proc. EUSIPCO*, 2010, pp. 532–536.

[22] *Digital Network Echo Cancellers*, ITU-T Rec. G.168, 2002.

[23] J. Benesty, C. Paleologu, and S. Ciochină, "On regularization in adaptive filtering," *IEEE Trans. Audio, Speech, Language Processing*, to appear.